Effective Dynamic Voltage Scaling through Accurate Performance Modeling *

Chung-Hsing Hsu and Wu-chun Feng
{chunghsu,feng}@lanl.gov
Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545

Keywords: Power-aware computing, dynamic voltage scaling, interval-based voltage scheduling, performance modeling, power-performance tradeoff.

Abstract

Dynamic voltage scaling (DVS) is widely recognized as an effective way to reduce high CPU power consumption. The technique trades CPU performance for power reduction and energy savings. As a result, there have been many proposals on how to effectively manage a DVS processor to minimize the CPU power consumption while keeping the performance degradation within an acceptable range. Most of these proposals use a simple performance-prediction model which assumes that the execution time will double if the CPU speed is cut in half. Unfortunately, this model overly exaggerates the impact that the CPU speed has on the execution time. It is only in the worst case that the execution time doubles when the CPU speed in halved. In general, the actual execution time is less than double. In addition to addressing the importance of an accurate performance model, we develop such a model and formulate an optimal DVS schedule, which serves as the basis of a new interval-based DVS algorithm. We then evaluate the effectiveness of the algorithm via physical measurements on a notebook computer.

*This work was supported by the DOE LDRD Program through Los Alamos National Laboratory contract W-7405-ENG-36. Available as technical report LA-UR-03-7582.

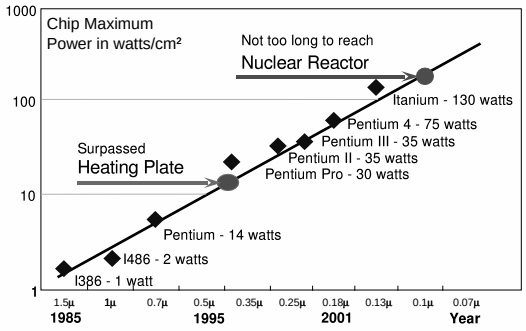

Figure 1: The power density of Intel CPUs.

1 Introduction

The power consumption of a CPU continues to follow Moore's Law by doubling every 18 months. Equally alarming is the growth of the power density, as shown in Figure 1. Because high power consumption raises temperature, reduces reliability, and increases the costs of thermal packaging, power delivery, and cooling, dynamic voltage scaling (DVS) has been proposed as a way to address these issues. DVS allows the CPU supply voltage to be adjusted at run time. This ability to dynamically adjust voltage for power savings is critical as CPU power consumption is directly proportional to the square of voltage. Since reducing the voltage may also require the CPU to be reduced, performance with respect to speed (or execution time) degrades. In short, DVS trades off performance for power reduction.

Many algorithms have been proposed to effectively manage a DVS processor by minimizing its power consumption within some performance-loss bounds. In general, a DVS algorithm needs to

determine *when* to adjust the current frequency-voltage setting (*scaling points*) and to *which* new frequency-voltage setting (*scaling factors*). For example, current interval-based DVS algorithms (e.g. [23, 18]) set the scaling points at the beginning of each time interval and determine the scaling factors by predicting the upcoming CPU workload.

To calculate the scaling factors, a performance model relating application performance to the CPU speed is required. Many DVS algorithms assume that the performance of an application scales proportionally to the CPU speed, i.e.,

$$T(f) \quad = \quad (1/f) \cdot W$$

where $T(f)$ (in seconds) is the execution time of a task running at frequency f, and W (in cycles) is the amount of the CPU work to be done. In other words, this model predicts that the performance will be improved by two-fold when the CPU speed is twice as fast.

Unfortunately, the above simple model ignores two important issues. First, the CPU work requirement W of a task is often unpredictable in realistic systems [18]. Second, W is not always independent of the frequency f [37, 30]. It is well-known that the performance of the memory- or I/O-bound applications cannot be improved in parallel with the CPU speed. In fact, Predtechenski [26] observed that the majority of application-class benchmarks exhibit the following relationship between the execution time $T(f)$ and the frequency f.

$$T(f) \quad = \quad (1/f) \cdot c_1 + c_0 \tag{1}$$

where c_0 and c_1 are two constants. A simple calculation reveals that the number of CPU cycles W is indeed influenced by the frequency.

$$W = f \cdot T(f) = c_1 + c_0 \cdot f$$

This paper addresses the aforementioned problems in the following way. We assume that a *constant* CPU work requirement still exists regardless of the operating CPU frequency; it may not be in terms of CPU cycles, but it is definitely embedded in an accurate performance-prediction model $T(f)$. Given such a performance model, we formulate a DVS scheduling problem and characterize its optimal solution. Based on the optimal schedule, we propose a new DVS algorithm that takes into account the performance behavior of a program, and results in a solution that is more effective than previous DVS algorithms. The main contributions of this paper include the following.

- **A theoretical result on the optimal DVS schedule in terms of our new performance model.** The significance of the theoretical result is that it can also be applied to a DVS processor that *does not* strictly obey the following relationship between frequency f and voltage V,

  $$f = k \cdot (V - V_T)^\alpha / V \tag{2}$$

  where k, V_T, and α are constants, $1 \leq \alpha \leq 2$ and $V_T \ll V$. Though many DVS algorithms derive their schedules based on this relationship, it is seldom established in real DVS processors due to the limited set of frequency and voltage settings.

- **The design of an interval-based DVS algorithm based on the theoretical result.** In the algorithm, the performance model is abstracted as a single parameter called the *performance-scalability factor*. Conceptually, it is similar to the scalability of performance in the field of parallel processing, but here the number of processors is replaced by various CPU frequencies.

- **The evaluation of the algorithm through physical measurements on a laptop.** Five popular DVS algorithms are compared for a representative set of `SPEC95` benchmarks. The experimental results show that an accurate performance model is very important for an effective DVS algorithm and that our new DVS algorithm is the most effective among the implemented DVS algorithms.

The rest of the paper is organized as follows: Section 2 overviews some of the previous theoretical results on the optimal DVS schedule. Section 3 presents the new theoretical result in term of performance model and proposes a new DVS algorithm based on the result. The experimental results of the algorithm are presented in Section 4, followed by more related work in Section 5 and future work in Sections 6.

2 Background

A typical DVS system consists of frequency and voltage combinations at which a processor is allowed to operate. We refer to each such combination as a frequency-voltage pair (V, f) or a performance level that is indexed by the frequency

$f \in [f_{min}, f_{max}]$. At each performance level f, the system power consumption is denoted as P_f. For a DVS system that only provides n performance levels f_i, we assume $f_1 < \cdots < f_n$. Finally, as with previous work, we assume that the performance model $T(f)$ is non-increasing; that is, a task running at a higher frequency will finish faster.

The essence of many DVS algorithms can be described as solving the following minimization problem: given a task of workload W cycles and deadline D seconds, find out a schedule $\{ t_f \}$ such that when the task is executed for t_f seconds at frequency f, the total energy usage E is minimized, the deadline D is met, and the required work W is performed.

$$\min E = \sum_f P_f \cdot t_f \tag{3}$$

subject to

$$\sum_f t_f \le D \tag{4}$$
$$\sum_f f \cdot t_f = W \tag{5}$$
$$t_f \ge 0 \tag{6}$$

Without loss of generality, we assume $D \ge \min_f W/f$; otherwise, the problem has no feasible solution.

Previous theoretical results on the characterization of the optimal solution $\{t_f^*\}$ rely heavily on $P_f = P(f)$ where $P(f)$ is a convex function. For example, Yao et al. [38] showed that the optimal schedule is

$$t_f^* = \begin{cases} D & \text{if } f = W/D \\ 0 & \text{otherwise} \end{cases}$$

In other words, running the task at a single frequency and finishing the task execution right at the deadline will minimize the total energy usage. This result can be applied to a DVS system with an infinite number of performance levels if $P(f)$ is convex on $[0, \infty]$ and $P(0) = 0$. For a system with only a few performance levels available, Ishihara and Yasuura [15] showed that scheduling with at most two frequencies, neighboring to W/D, will minimize the energy consumption. Qu [27] put it more concretely that the optimal schedule $\{t_f^*\}$ is

$$t_f^* = \begin{cases} \frac{f_{j+1} \cdot D - W}{f_{j+1} - f_j} & f = f_j \\ D - t_{f_j}^* & f = f_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

where $f_j \le W/D < f_{j+1}$. It should not be hard to see why this works since

$$P'(f) = P(f_{i-1}) \cdot \frac{f_i - f}{f_i - f_{i-1}} + P(f_i) \cdot \frac{f - f_{i-1}}{f_i - f_{i-1}}$$

for $f_{i-1} \le f < f_{i+1}$ is indeed convex [16].

Saewong and Rajkumar [29] extended the above results for a more realistic case of $P(f)$ convex on $[f_{min}, f_{max}]$. The new result says that running the task at a single frequency will still minimize the total energy usage, and this optimal frequency f^* can be computed using the following equation.

$$f^* = \arg \min_{f \ge W/D} P(f)/f$$

The aforementioned result of $f^* = W/D$ then becomes a special case of this extended result since $P(f)/f$ is nondecreasing when $P(f)$ is convex on $[0, \infty]$ and $P(0) = 0$, In fact, the result will hold as long as

$$\frac{P_{f_2} - P_{f_1}}{f_2 - f_1} \le \frac{P_{f_3} - P_{f_2}}{f_3 - f_2} \le \cdots \le \frac{P_{f_n} - P_{f_{n-1}}}{f_n - f_{n-1}}$$

Note that the condition can be removed when $D \ge \max_f W/f$ since the problem becomes the classical fractional knapsack problem.

The theoretical results mentioned above are extensively used by many DVS algorithms. Unfortunately, these results all depend on the two assumptions: W is known a priori and $T(f) = W/f$. The performance model $T(f) = W/f$ does not work well for the memory- and I/O-bound applications [14, 37, 30]. Furthermore, it is parameterized in the CPU work requirement W that is often unpredictable in realistic systems [18]. In the next section, we will present a methodology that computes W and $T(f)$ implicitly and appropriately models the performance of memory- and I/O-bound applications relative to f.

3 Theoretical Results

In order to eliminate the parameter W, we assume that the total work of a task is evenly distributed over time and replace Equation (5) with the following:

$$\sum_f \frac{t_f}{T(f)} = 1$$

To simplify the discussion, instead of solving for the variables t_f, the new problem solves a scaled

version of variables, $r_f = t_f/T(f)$. The complete formulation is listed as follows.

$$\min \sum_f E_f \cdot r_f \tag{7}$$

subject to

$$\sum_f T(f) \cdot r_f \leq D \tag{8}$$

$$\sum_f r_f = 1 \tag{9}$$

$$r_f \geq 0 \tag{10}$$

where $E_f = P_f \cdot T(f)$ is the total energy usage of a task running at frequency f. Again, without loss of generality, we assume that $D \geq \min_f T(f)$. The optimal solution $\{r_f^*\}$ of the new problem can be characterized by the following theorems. Due to the space limitation, we leave all proofs to the technical report.

Theorem 3.1 (Unconstrained Scheduling for Energy Minimization) *For $D \geq \max_f T(f)$, the optimal solution $\{r_f^*\}$ is*

$$r_f^* = \begin{cases} 1 & f = f^* \\ 0 & \text{otherwise} \end{cases}$$

where $f^* = \arg\min_f E_f$.

Theorem 3.2 (Deadline-Constrained Scheduling for Energy Minimization) *For $D < \max_f T(f)$, if the ratios $\gamma_i = \frac{E(f_i)-E(f_{i+1})}{T(f_i)-T(f_{i+1})}$ are negative and non-increasing, the optimal solution $\{r_f^*\}$ is*

$$r_f^* = \begin{cases} \frac{D-T(f_{j+1})}{T(f_j)-T(f_{j+1})} & f = f_j \\ 1 - r_{f_j}^* & f = f_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

where $T(f_{j+1}) < D \leq T(f_j)$.

Theorem 3.1 presents an uninteresting case that when the deadline is sufficiently large, choosing among all performance levels the level that minimizes the total energy usage is the best strategy. For the case of $D < \max_f T(f)$, *Theorem 3.2* basically says that if $E(f)$ is a convex, non-increasing function of $T(f)$, then running at the ideal single frequency f^*, where $T(f^*) = D$, will minimize the total energy usage. If the frequency f^* is not directly supported by the system, the two neighboring frequencies f_j and f_{j+1}, $T(f_{j+1}) < D < T(f_j)$, can emulate it and results in the minimum energy consumption.

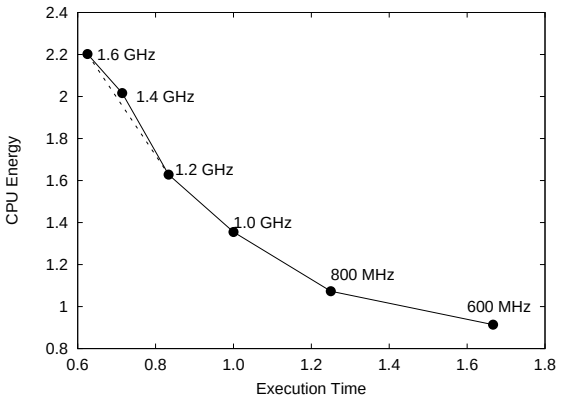

Figure 2: The execution time and CPU energy consumption of a 600-1600 MHz Intel Pentium M processor with six performance levels.

To appreciate *Theorem 3.2*, let us consider a 600-1600 MHz Intel Pentium M processor with the performance levels as specified in its datasheet [5]. For the moment, we assume $E(f) = V^2$ and $T(f) = 1/f$. The constants are removed since we only look into the convexity of $E(T(f))$. Figure 2 depicts the convexity of $E(T(f))$, where all the performance levels except at 1.4 GHz will satisfy the condition of *Theorem 3.2*. In fact, the level at 1.4 GHz should never be used in any DVS algorithm because its speed can be emulated by other levels with lower energy consumption [19, 21]. Specifically, 1.4 GHz can be emulated by running half of the time at 1.2 GHz and half the time at 1.6 GHz. This consumes (15 W + 24.5 W)/2 = 19.75 W while the 1.4-GHz setting consumes 20 W. In this paper, we assume that our DVS system does not contain these energy-inefficient levels.

Based on *Theorem 3.2*, we propose a new interval-based DVS algorithm *beta*. In developing this new algorithm, we first abstract the performance model as a single parameter called the *performance-scalability factor* $\beta \in [0, 1]$, i.e.,

$$\frac{T(f)}{T(f_{max})} = \beta \cdot \frac{f_{max}}{f} + (1 - \beta) \tag{11}$$

The parameter β indicates the sensitivity of the application performance to the change in CPU speed. If $\beta = 1$, that means the execution time will be cut in half when the CPU speed is twice as fast. If $\beta = 0$, the execution time will remain constant even running at the slowest frequency. In terms of Equation

1. Compute the ideal single frequency f^*.

$$f^* = \begin{cases} f_{min} & \text{if } \beta \leq \delta \\ f_{max}/(1+\delta/\beta) & \text{otherwise} \end{cases}$$

2. Figure out f_j and f_{j+1}.

$$f_j \leq f^* < f_{j+1}$$

3. Compute the ratios, r_{f_j} and $r_{f_{j+1}}$.

$$r_{f_j} = \frac{(1+\delta/\beta)/f_{max} - 1/f_{j+1}}{1/f_j - 1/f_{j+1}}$$
$$r_{f_{j+1}} = 1 - r_{f_j}$$

4. For every second, run r_{f_j} at frequency f_j and run $r_{f_{j+1}}$ at frequency f_{j+1}.

Figure 3: The new DVS algorithm *beta*. The quantity δ specifies the performance-loss bound in terms of the relative execution time, i.e., $D = (1+\delta) \cdot T(f_{max})$.

(1), β =1 if $T(f) = c_1/f$, and $\beta = 0$ if $T(f) = c_0$. The new DVS algorithm *beta* is shown in Figure 3.

f (MHz)	V
1067	1.15
1333	1.25
1467	1.30
1600	1.35
1800	1.45

Table 1: The five energy efficient performance levels in the HP NX9005 laptop.

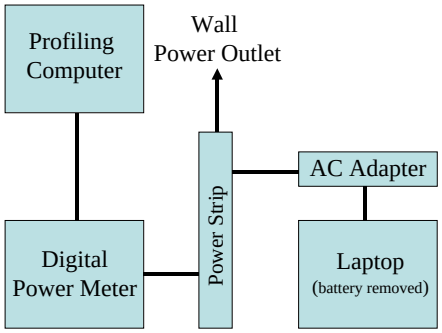

Figure 4: The experimental setup.

4 A Case Study

In this section, we describe our experimental environment in which we evaluate the *beta* DVS algorithm and present experimental results.

4.1 Hardware Platform

The hardware platform in our experiments is an HP NX9005 notebook computer. This computer includes a mobile AMD Athlon XP 2200+ processor with a 256-KB level-two cache, 256-MB DDR SDRAM, 266-MHz front-side bus, a 30-GB hard disk, and a 15-inch TFT LCD display. The processor supports DVS under software control by writing the desired frequency and voltage to the machine-specific registers in the AMD Athlon XP. The five performance levels for the experiments are shown in Table 1. The transition time from one level to another level is set as 30 microseconds. Through the measurements, we found that the entire laptop consumes 29.2 – 50.8 watts when idle, and 33.8 – 65.8 watts when running a CPU-intensive benchmark.

Figure 4 shows the experimental setup. The experimental results were collected through a Yokogawa WT210 digital power meter [12]. The power meter continuously samples the instantaneous wattage sampled at a rate of 50 kHz (i.e., every 20 μs). The laptop runs the Linux 2.4.18 kernel. All the benchmarks were compiled by GNU compilers with optimization level `-O6`. All the benchmarks were run to completion; each run took over a minute. During the measurements, the battery was removed and the monitor was turned off.

A few representative `SPEC95` benchmarks were used for the experiments. The `SPEC` benchmarks [6] emphasize the performance of the CPU and memory, but not other computer components such as I/O (disk drives), networking or graphics. In this paper, we chose to use `SPEC` benchmarks because they demonstrate a range of sensitivity to the CPU frequency change, as shown in Figure 5. The β value for each benchmark was derived by profiling $T(f)$ for all the performance levels on the laptop and estimating using the least-square fitting on Equation (11).

program	β
swim	0.02
tomcatv	0.16
hydro2d	0.19
su2cor	0.27
applu	0.34
apsi	0.37
mgrid	0.51
wave5	0.52
turb3d	0.79
fpppp	1.00

program	β
compress	0.37
vortex	0.65
gcc	0.82
ijpeg	0.95
perl	0.99
m88ksim	0.99
go	1.00
li	1.00

Figure 5: The entire `SPEC95` benchmark suite and the corresponding β value for each benchmark on the HP NX9005 notebook computer.

4.2 Software Platform

To illustrate the importance of an accurate performance model in an effective DVS algorithm, we compared our algorithm *beta* with a few other interval-based implementations. The experiments by no means represent a comprehensive comparison among all existing approaches. Nevertheless, we feel that it illustrates our point well enough. The following is a brief description of each algorithm we tested.

opt This algorithm assumes E_f and $T(f)$ for each frequency f are known a priori. It uses a LP solver to solve the minimization problem in Section 3 to generate the optimal DVS schedule $\{r_f^*\}$ and executes the program using the schedule. In other words, algorithm *opt* first executes the program at frequency f_1 for $r_{f_1}^* \cdot T(f_1)$ seconds, and then at frequency f_2 for $r_{f_2}^* \cdot T(f_2)$ seconds, and so on. According to *Theorem 3.2*, at most two r_f^*'s have nonzero values; that is, algorithm *opt* algorithm will adjust the performance level at most once during the entire program execution.

2step This algorithm mimics the policy used in Intel's `SpeedStep` technology. The algorithm assumes dual speeds in the system. It monitors the CPU utilization rate (how long the CPU is active in an interval) in each interval. If the CPU utilization rate is higher than a pre-defined threshold level, the algorithm will set the CPU to a fast speed; if it is lower than the other pre-defined threshold level, the CPU will be set to a low speed. In our implementation, these two threshold levels are 50% and 10%, respectively.

freq This algorithm is similar to strategies that reclaim the slack time between the actual processing time and the worst-case execution time (e.g., [17, 22, 1, 2]). Specifically, the algorithm keeps track of the amount of work remaining W_{left} and the time left before the deadline T_{left}. The desired frequency is then computed through $f = W_{left}/T_{left}$. The algorithm is most effective when there is a significant amount of the slack time, usually coming from the over-estimation of the worst-case execution time.

mips This algorithm represents a strategy that is guided by an externally specified performance metric. Specifically, the new frequency f_{new} is computed through

$$f_{new} = f_{prev} \cdot \texttt{MIPS}_{target}/\texttt{MIPS}_{observed}$$

where f_{prev} is the operating frequency for the previous interval, $\texttt{MIPS}_{target}$ is the externally specified performance requirement, and $\texttt{MIPS}_{observed}$ is the real `MIPS` rate in the previous interval. In our experiments, each benchmark has its own $\texttt{MIPS}_{target}$, which is derived by measuring the MIPS rate for the entire application and then dividing it by $(1 + \delta)$. Algorithms in this category include [9, 4].

4.3 Experimental Results

Table 2 presents the experimental results across all five aforementioned DVS algorithms. In general, when a program is CPU intensive (β close to 1), there is little opportunity in reducing a significant amount of energy within a tight performance bound. The more appropriate strategy for this case is through parallelism on multiple DVS processors.

Second, it can be seen that algorithm *opt* and *beta* are on average more effective than algorithm *mips*. On the other hand, algorithms *2step* and *freq* have no effect at all. Among all the tested algorithms, algorithm *2step* is the only one that does not need any external information. All other algorithms have different abstractions of the performance model. The experimental results suggest that an accurate performance model is required for the DVS algorithm to be effective.

Algorithm *2step* has two tunable threshold values. However, it is an open question how to do this

program	β	*opt*	*2step*	*freq*	*mips*	*beta*
swim	0.02	1.00/0.57	1.00/1.00	1.00/0.95	1.00/0.79	1.00/0.57
tomcatv	0.16	1.04/0.72	1.00/1.01	0.99/0.95	1.02/0.99	1.05/0.75
su2cor	0.27	1.03/0.79	0.98/0.99	1.00/0.97	1.01/0.97	1.03/0.81
compress	0.37	1.03/0.88	1.00/1.00	1.00/0.95	1.03/0.89	1.04/0.85
mgrid	0.51	1.03/0.90	0.99/0.99	1.01/0.99	1.00/1.00	1.02/0.94
vortex	0.65	1.04/0.93	0.99/0.99	1.00/0.96	1.05/0.92	1.02/0.95
turb3d	0.79	1.04/0.94	1.00/1.00	1.02/0.96	1.00/1.00	1.03/0.97
go	1.00	1.05/0.96	1.00/0.99	1.02/0.98	1.00/1.00	1.03/0.97

Table 2: The effectiveness of the five different DVS algorithms. Each table entry is in the format of *relative-time/relative-energy* with respect to the total execution time and system energy usage when running the application at the highest performance level throughout the entire execution.

in a systematic fashion. Grunwald et al. [11] found that these threshold values are application dependent and data sensitive. In fact, the algorithm will *not* have an appropriate threshold setting that can make it as effective as *beta* or *mips*. This is because the CPU for non-interactive applications is active almost all the time.

Algorithm *freq* uses the number of CPU cycles as the metric for specifying the CPU work requirement. However, the number of CPU cycles varies significantly across performance levels. Since this number is usually the largest at the highest performance level, algorithm *freq* exaggerates the amount of the CPU work to be done and results in less effective energy reduction. As a result, algorithm *freq* is more appropriate for CPU-intensive applications only.

Algorithm *mips* performs better than algorithm *freq*. This is because the number of retired instructions is a better metric for specifying the CPU work requirement. For the benchmarks we tested, the number of instructions tends to remain constant across all performance levels. Unfortunately, the MIPS rate varies significantly as well, especially for CPU-intensive applications.

Finally, algorithm *opt* requires the total execution time to be known a priori. It is therefore the most appropriate to use the algorithm in special-purpose computing systems. On the other hand, our new *beta* algorithm only relies on the abstract performance model β and the performance-loss bound δ. The effectiveness of the algorithm is determined by how accurately the β value can capture the performance impact with respect to the CPU frequency.

Note that β accurately captures the performance impact if Equation (1) holds. By leveraging the observation that the number of instructions remains constant across all performance levels, one can replace execution time with the reciprocal of the MIPS (million instructions per second) rate to faciliate the derivation of β on the fly (i.e., without profile runs) through linear regression on frequency-varying MIPS rates.

5 Related Work

In this paper we focused on DVS algorithms for a single application. The DVS algorithms for a set of tasks have also been studied extensively, especially in the context of real-time systems. In terms of the optimal DVS schedule for a a task set, Yao et al. [38] provided a polynomial-time algorithm for a set of independent tasks when preemption is allowed and the task priority is not fixed. The algorithm is optimal on a DVS system with infinite performance levels if $P(f)$ is convex and there is no transition overhead. Kwon and Kim [16] extended the algorithm to a DVS system with a few performance levels. Yun and Kim [39] showed that if the task priority is fixed, then the problem becomes NP-hard; that is, no optimal polynomial-time algorithm is likely to exist. As illustrated by Quan and Hu [28], the problem is more difficult to solve because the preemption relationship among the tasks is much more complex to analyze. The problem is NP-complete if the underlying system can vary the voltage within a maximum rate when the task priority is not fixed [27].

There are many proposed DVS algorithms in the literature. Due to the space limitation, we only discuss work related to interval-based DVS algorithms

for general-purpose computer systems. For DVS in real-time systems, Unsal and Koren provide a recent survey [33]. Weiser et al. [35] first presented the idea of interval-based DVS algorithms for a general-purpose operating system. Govil et al. [10] and Lorch and Smith [18] extended the work and considered a large number of different workload-prediction and speed-selection policies. Some algorithms were examined by Pering et al. [23] through trace-driven simulation and by Grunwald et al. [11] through physical measurements. Both studies show that these algorithms result in unsatisfactory performance degradation.

There are at least two reasons why these interval-based algorithms are not able to hide the performance penalty. First, future workloads are sometimes irregular and thus unpredictable. To tackle this problem, a group of researchers (e.g., [20, 32, 34]) applied a bit of control theory to design new DVS algorithms. For example, Sinha and Chandrakasan [32] proposed to use an adaptive filtering technique to "smooth out" the noises in the past workload profile. Pouwelse et al. [25] argued that the application itself can provide some useful information to the DVS algorithm. For example, Ghiasi et al. [9] and Childers et al. [4] suggested to use externally provided, desired IPC rate to direct the DVS algorithms. Hsu and Kremer [13] used the compiler techniques to embed the scaling points and factors in the executable. Others [2, 22, 31] associated compiler-derived scaling points with a piece of code that can calculates the scaling factors at run time.

Second, the optimal interval length is system and application dependent. Chandrasena et al. [3] proposed to buffer interval results until the scaling factor matches well with the system's available frequencies. That is, the scaling points are at the beginning of intervals but not every one of them. The other proposal is to detect changes or episodes in the aggregate behavior [8, 20].

6 Conclusions and Future Work

In this paper we have presented a DVS scheduling problem that incorporates a parametric performance model and that does not depend on the explicit CPU work requirement. The optimal solution for the scheduling problem was characterized. The significance of the optimality theorem is that the theorem does not rely on a specific relationship between frequency and voltage in a performance level that many previous DVS algorithms assume. Based on the characterization of the optimal schedule, a new interval-based DVS algorithm *beta* was proposed. In the algorithm, the performance model is abstracted as a single parameter β. The algorithm was compared against a few other interval-based DVS algorithms through the physical measurements on a laptop. The experimental results show that an accurate performance model is indeed very important.

We plan to study whether we can estimate the abstracted performance model β reasonably accurately on the fly for a large class of $T(f)$. One possible way is to use the techniques proposed by Lorch and Smith [18] in estimating the CPU work requirement at run time. Since the number of CPU cycles is no longer a good metric for this purpose, we plan to investigate whether the number of instructions will be a better candidate. We also would like to compare our DVS algorithm *beta* with a few more implementations such as [7, 13, 19, 20, 24, 36]. Finally, we are also looking into applying our new DVS algorithm to the high-performance computing platforms for a better energy efficiency and thermal management.

References

[1] N. AbouGhazaleh, D. Mossé, B. Childers, and R. Melhem. Toward the placement of power management points in real time applications. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP)*, September 2001.

[2] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework. In *Proceedings of Design, Automation and Test in Europe Conference (DATE)*, March 2002.

[3] L. Chandrasena, P. Chandrasena, and M. Liebelt. An energy efficient rate selection algorithm for quantized dynamic supply voltage scaling. In *Proceedings of the International Symposium on Systems Synthesis (ISSS)*, October 2001.

[4] B. Childers, H. Tang, and R. Melhem. Adapting processor supply voltage to instruction-level parallelism. In *Kool Chips Workshop*, December 2000.

[5] Intel Corporation. Intel Pentium M processor datasheet. Publication 252612-002, June 2003.

[6] The Standard Performance Evaluation Corporation. http://www.spec.org.

[7] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[8] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, July 2001.

[9] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity Effective Design*, June 2000.

[10] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *the 1st ACM International Conference on Mobile Computing and Networking (MOBICOM-95)*, pages 13–25, November 1995.

[11] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI)*, October 2000.

[12] N. Hirofumi, N. Naoya, and T. Katsuya. WT210/WT230 digital power meters. Yokogawa Technical Report 35, 2003.

[13] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI)*, June 2003.

[14] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic frequency and voltage scheduling. In *Workshop on Power-Aware Computer Systems (PACS)*, November 2000.

[15] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 197–202, August 1998.

[16] W-C Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Proceedings of the 40th Design Automation Conference (DAC)*, June 2003.

[17] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proceedings of the 37th Conference on Design Automation (DAC)*, June 2000.

[18] J. Lorch and A. Smith. Improving dynamic voltage algorithms with PACE. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, June 2001.

[19] J. Lorch and A. Smith. Operating system modifications for task-based speed and voltage scheduling. In *The First International Conference on Mobile Systems, Applications, and Services (MobiSys)*, May 2003.

[20] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, October 2002.

[21] A. Miyoshi, C. Lefurgy, E. Hensbergen, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing (ICS)*, June 2002.

[22] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compiler and Operating Systems for Low Power (COLP)*, October 2000.

[23] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of 1998 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 76–81, August 1998.

[24] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Symposium on Operating Systems Principles (OSDI)*, October 2001.

[25] J. Pouwelse, K. Langendoen, and H. Sips. Application-directed voltage scaling. *IEEE Transactions on VLSI Systems*.

[26] A. Predtechenski. A method for benchmarks analysis. In *Workshop on Performance Evaluation with Realistic Applications*, January 1999.

[27] G. Qu. What is the limit of energy saving by dynamic voltage scaling? In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, November 2001.

[28] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the 38th Design Automation Conference (DAC)*, June 2001.

[29] S. Saewong and R. Rajkumar. Practical voltage-scaling for fixed-priority RT-systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposiuam (RTAS)*, May 2003.

[30] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-aware static timing analysis. In *The 24th IEEE International Real-Time Systems Symposium (RTSS)*, December 2003.

[31] D. Shin and J. Kim. A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01)*, August 2001.

[32] A. Sinha and A. Chandrakasan. Dynamic voltage scheduling using adaptive filtering of workload traces. In *Proceedings of the 14th International Conference on VLSI Design*, January 2001.

[33] O. Unsal and I. Koren. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE*, 91(7), July 2003.

[34] A. Varma, B. Ganesh, M. Sen, S. Choudhary, L. Srinivasan, and B. Jacob. A control-theoretic approach to dynamic voltage scaling. In *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, October 2003.

[35] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, pages 13–23, November 1994.

[36] Andreas Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, August 2002.

[37] F. Xie, M. Martonosi, and S. Malik. Compile time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation (PLDI)*, June 2003.

[38] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 374–382, October 1995.

[39] H-S Yun and J. Kim. On energy-optimal voltage scheduling or fixed-priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3), August 2003.